Reconfigurable processor array exploiting ILP and TLP

TECHNICAL FIELD

The technical field of this invention is processor architectures, particularly related to multi-processor systems, methods for programming said processors and compilers for implementing said methods.

BACKGROUND ART

A Very Long Instruction Word (VLIW) processor is capable of executing many operations within one clock cycle. Generally, a compiler reduces program instructions into basic operations that the processor can perform simultaneously. The operations to be performed simultaneously are combined into a very long instruction word (VLIW). The instruction decoder of the VLIW processor decodes and issues the basic operations comprised in a VLIW each to a respective processor data-path element. Alternatively, the VLIW processor has no instruction decoder, and the operations comprised in a VLIW are directly issued each to a respective processor data-path element. Subsequently, these processor data-path elements execute the operations in the VLIW in parallel. This kind of parallelism, also referred to as instruction level parallelism (ILP), is particularly suitable for applications which involve a large amount of identical calculations, as can be found e.g. in media processing. Other applications comprising more control-oriented operations, e.g. for servo control purposes, are not suitable for programming as a VLIW program. However, often these kinds of programs can be reduced to a plurality of program threads that can be executed independently of each other. The execution of such threads in parallel is also denoted as thread-level parallelism (TLP). A VLIW processor is, however, not suitable for executing a program using thread-level parallelism. Exploiting the latter type of parallelism requires that sub-sets of processor data-path elements have an independent control flow, i.e. that they can access their own programs in a sequence independent of each other, e.g. they are capable of independently performing conditional branches. The data-path elements in a VLIW processor, however, all execute a sequence of instructions in the same order. The VLIW processor can, therefore, only execute one thread.

To control the operations in the data pipeline of a VLIW processor, two different mechanisms are commonly used: data-stationary and time-stationary. In the case of data-stationary encoding, every instruction that is part of the processor's instruction-set controls a complete sequence of operations that have to be executed on a specific data item, as it traverses the data pipeline. Once the instruction has been fetched from program memory and decoded, the processor controller hardware will make sure that the composing operations are executed in the correct machine cycle. In the case of time-stationary coding, every instruction that is part of the processor's instruction-set controls a complete set of operations that have to be executed in a single machine cycle. These operations may be applied to several different data items traversing the data pipeline. In this case it is the responsibility of the programmer or compiler to set up and maintain the data pipeline. The resulting pipeline schedule is fully visible in the machine code program. Time-stationary encoding is often used in application-specific processors, since it saves the overhead of hardware necessary for delaying the control information present in the instructions, at the expense of larger code size.

DISCLOSURE OF THE INVENTION

It is an object of the invention to provide a processor that is capable of exploiting both instruction-level parallelism as thread-level parallelism or a combination thereof, during execution of an application.

For that purpose, a processor according to the invention comprises a plurality of processing elements, the plurality of processing elements comprising a first set of processing elements and at least a second set of processing elements;
wherein each processing element of the first set comprises a register file and at least one instruction issue slot, the instruction issue slot comprising at least one functional unit, and the processing element being arranged to execute instructions under a common thread of control;
wherein each processing element of the second set comprises a register file and a plurality of instruction issue slots, each instruction issue slot comprising at least one functional unit, and the processing element being arranged to execute instructions under a common thread of control;
and wherein the number of instruction issue slots in the processing elements of the second set is substantially higher than the number of instruction issue slots in the processing elements of the first set;

and wherein the processing system further comprises inter-processor communication means arranged for communicating between processing elements of the plurality of processing elements. The computation means can comprise adders, multipliers, means for performing logical operations, e.g. AND, OR, XOR etc., lookup table operations, memory accesses, etc.

A processor according to the present invention allows exploiting both instruction-level parallelism and thread-level parallelism in an application, and a combination thereof. In case a program has a large degree of instruction-level parallelism, the application can be mapped onto one or more processing elements of the second set of processing elements. These processing elements have multiple issue slots allowing the execution of multiple instructions in parallel under one thread of control, and are therefore suited for exploiting instruction-level parallelism. If a program has a large degree of thread-level parallelism, but a low degree of instruction-level parallelism, the application can be mapped onto the processing elements of the first set of processing elements. These processing elements have a relatively lower number of issue slots allowing the mostly sequential execution of a series of instructions under one thread of control. By mapping each thread on such a processing element, several threads of control can be present in parallel. In case a program has a large degree of thread-level parallelism, and one or more threads have a large degree of instruction-level parallelism, the application can be mapped onto a combination of processing elements of the first set as well the second set of processing elements. Processing elements of the first set allow execution of threads consisting of a mostly sequential series of instructions, while processing elements of the second set allow execution of threads having instructions that can be executed in parallel. As a result, the processor according to the invention can exploit both instruction-level parallelism as well as thread-level parallelism, depending on the type of application that has to be executed.

"Architecture and Implementation of a VLIW Supercomputer" by Colwell et al., in Proc. of Supercomputing 1990, p.p. 910 – 919, describe a VLIW processor, which can either be configured as two 14-operations-wide processor, each independently controlled by a respective controller, or one 28-operations-wide processor controlled by one controller. EP0962856 discloses a Very Large Instruction Word processor, including plural program counters, and is selectively operable in either a first or a second mode. In the first mode, the data processor executes a single instruction stream. In the second mode, the data processor executes two independent program instruction streams simultaneously. Said documents,

however, do neither disclose the principle of a processor array with a number of processing elements executing threads in parallel, said threads varying from having no instruction-level parallelism to a large degree of instruction-level parallelism, nor does it disclose how such a processor array could be realized.

An embodiment of the invention is characterized in that the processing elements of the plurality of processing elements are arranged in a network, wherein a processing element of the first set is arranged for direct communication with a processing element of only the second set, via the inter-processor communication means; and wherein a processing element of the second set is arranged for direct communication with a processing element of only the first set, via the inter-processor communication means. In practical applications, functions that have a large degree of instruction-level parallelism and functions having a low degree of instruction-level parallelism will be interleaved. By choosing an architecture in which processing elements of the first type and second type are interleaved as well, an efficient mapping of the application onto the processing system is allowed.

An embodiment of the invention is characterized in that the inter-processor communication means comprise a data-driven synchronized communication means. By using a data-driven synchronization mechanism to govern communication across the processing elements, it can be guaranteed that no data is lost during communication.

An embodiment of the invention is characterized in that the processing elements of the plurality of processing elements are arranged to be bypassed by the inter-processor communication means. An advantage of this embodiment is that it increases the flexibility of mapping the application onto the processing system. Depending on the degree of instruction-level parallelism as well as task-level parallelism of the application, one or more processing elements may not be used during execution of the application.

Further embodiments of the invention are described in the dependent claims. According to the invention a method for programming said processing system, as well as a compiler program product being arranged for implementing all steps of said method for programming a processing system, when said compiler program product is run on a computer system, are claimed as well.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 shows a schematic diagram of a processing system according to the invention.

Figure 2 shows an example of a processing element of the second set of processing elements in more detail.

Figure 3 shows an example of a processing element of the first set of processing elements in more detail.

Figure 4 shows an example of the data-path connection between processing elements in more detail.

Figure 5 shows the application graph of an application to be executed by a processing system according to the invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

Figure 1 schematically shows a processing system according to the invention. The processing system comprises a plurality of processing elements PE1 – PE23, having a first set of processing elements PE1 – PE15, and a second set of processing elements PE17 – PE23. The processing elements can exchange data via data-path connections DPC. In the preferred embodiment shown in Figure 1, the processing elements are arranged such that between two processing elements of the first set, there is one processing element of the second set, and vice versa, and the data-path connections provide for data exchange between neighboring processing elements. Non-neighboring processing elements may exchange data by transferring it via a chain of mutually neighboring processing elements. Alternatively, or in addition, the processor system may comprise one or more global busses spanning sub-sets of the plurality of processing elements, or point-to-point connections between any pair of processing elements. Alternatively, the processing system may comprise more or less processing elements, or more than two different sets of processing elements, such that processing elements in the different sets comprise different numbers of issue slots, therefore supporting different levels of instruction-level parallelism per set.

Figure 2 shows an example of a processing element of the second set of processing elements PE17 – PE23 in more detail. Each processing element of the second set of processing elements comprises two or more issues slots (ISs), and one ore more register files (RFs), each issue slot comprising one or more functional units. The processing element in Figure 2 comprises five issue slots IS1-IS5, and six functional units: two arithmetic and

logic units (ALU), two multiply-accumulate units (MAC), an application-specific unit (ASU), and a load/store unit (LD/ST). The processing element also comprises five register files RF1-RF5. Issue slot S1 comprises two functional units: an ALU and a MAC. Functional units in a common issue slot share read ports from a register file and write ports to an interconnect network IN. In an alternative embodiment, a second interconnect network could be used in between register file and operation issue slots. The functional unit(s) in an issue slot have access to at least one register file associated with said issue slot. In Figure 2, there is one register file associated with each issue slot. Alternatively, more than one issue slot could be connected to a single register file. Yet another possibility is that multiple, independent register files are connected to a single issue slot, e.g. one different RF for each separate read port of a functional unit in the issue slot. The data path connections DPC between different processing elements are preferably driven from the load/store unit (LD/ST) in the respective processing element, so that communications across processing elements can be managed as memory transactions. Preferably, a different load/store unit (LD/ST) is used in association with the different data-path connections (DPC) connecting the processing element to other processing elements. This way, if the processing element is directly connected to e.g. four other processing elements, then four different load/store units are preferably used for communication with those processing elements, not shown in Figure 2. In addition, further load/store units could be added to the data-path of a processing element, and associated to data memories (e.g. RAM), either local to the processing element, or system-level memories, not shown in Figure 2. The functional units are controlled by a controller CT that has access to an instruction memory IM. A program counter PC determines the current instruction address in the instruction memory IM. The instruction pointed to by said current address is first loaded into an internal instruction register IR in the controller. The controller CT then controls data-path elements (functional units, register files, interconnect network) to perform the operations specified by the instruction stored in the instruction register IR. To do so, the controller communicates to the functional units via an opcode-bus OB, e.g. providing operation codes to the functional units, to the register files via an address-bus AB, e.g. providing addresses for reading and writing registers in the register file, and to the interconnect network IN through a routing-bus RB, e.g. providing routing information to the interconnect multiplexers. Processing elements of the second set comprise multiple issue slots, which allow exploiting instruction-level parallelism within a thread. For example, application functions with inherent instruction-level parallelism such as Fast Fourier

Transforms, Discrete Cosine Transforms and Finite Impulse Response filters can be mapped onto processing elements of the second set.

Figure 3 shows an example of a processing element of the first set of processing elements PE1 – PE15 in more detail. A processing element of the first set of processing elements comprises a relatively low number of issue slots, compared to processing elements of the second set of processing elements. A processing element of the first set further comprises one or more register files and a controller. The issue slots comprise one or more functional units, for example an arithmetic and logic unit, a multiply-accumulate unit or an application-specific unit. The processing element in Figure 3 comprises two issue slots IS6 and IS7 and two register files RF6 and RF7. Issue slot IS6 comprises two functional units: an ALU and a MAC. Functional units in a common issue slot share read ports from a register file and write ports to an interconnect network IN. Issue slot IS7 comprises a load/store unit (LD/ST) that drives the data-path connections (DPC) connecting the processing element with other processing element(s). Preferably, a different load/store unit (LD/ST) is used in association with the data-path connections (DPC) connecting the processing element directly to other processing elements. This way, if the processing element is directly connected to e.g. four other processing elements, then four different load/store units are preferably used for communication with those processing elements, not shown in Figure 3. In addition, further load/store (LD/ST) units could be added to the data-path of a processing element, and associated to data memories (e.g. RAM), either local to the processing element, or system-level memories, not shown in Figure 3. In an alternative embodiment, a second interconnect network could be used in between register file and operation issue slots. The functional unit(s) in an issue slot have access to at least one register file associated with said issue slot. In Figure 3, there is one register file associated with issue slot IS6, and another register file associated with issue slot IS7. Alternatively, independent register files are connected to the issue slot, e.g. one different RF for each separate read port of a functional unit in the issue slot. The functional units are controlled by a controller CT that has access to an instruction memory IM. A program counter PC determines the current instruction address in the instruction memory IM. The instruction pointed to by said current address is first loaded into an internal instruction register IR in the controller. The controller CT then controls data-path elements (functional units, register files, interconnect network) to perform the operations specified by the instruction stored in the instruction register IR. To do so, the controller communicates to the functional units via an opcode-bus OB, e.g. providing

operation codes to the functional units, to the register files via an address-bus AB, e.g. providing addresses for reading and writing registers in the register file, and to the interconnect network IN through a routing-bus RB, e.g. providing routing information to the interconnect multiplexers. Processing elements of the first set have a relatively lower number of issue slots and are therefore suitable for computing inherently sequential functions, for example Huffman coding.

Figure 4 shows an example of the data-path connection DPC between processing elements in more detail. In a preferred embodiment, the data-path connections use a data-driven synchronization mechanism, in order to prevent that data is lost during communication between processing elements. The data-path connection between processing elements PE2 and PE4, shown in Figure 4, comprises two blocking First-In-First-Out (FIFO) buffers BF. The FIFO buffers BF are controlled by control signals hold_w and hold_r. In case processing element PE2 or PE4 is trying to write data to a FIFO buffer BF that is full, the signal hold_w is activated, halting the entire processing element until another processing element reads at least one data element from that FIFO buffer, freeing up storage space in that FIFO buffer. In that case the hold_w signal is deactivated. A clock-gating mechanism can be used to halt the processing element from writing data to a full FIFO buffer, using the hold_w signal, as long as that FIFO buffer is full. In case a processing element PE2 or PE4 tries to read a value from a FIFO buffer that is empty, the hold_r signal is activated, halting the entire processing element until another processing element writes at least one data element into the FIFO buffer. At that moment the hold_r signal is deactivated and the processing element that was halted can start reading data from said FIFO buffer again. A clock-gating mechanism can be used to halt a processing element from reading data from an empty FIFO buffer, using the hold_r signal, as long as that FIFO buffer is empty.

In a preferred embodiment, processing elements in both sets are VLIW processors, wherein processing elements of the second set are wide VLIW processors, i.e. VLIW processors with many issue slots, while processing elements of the first set are narrow VLIW processors, i.e. VLIW processors with a small number of issue slots. In an alternative embodiment, processing elements of the second set are wide VLIW processors with many issue slots, and processing elements of the first set are single-issue slot Reduced Instruction Set Computer (RISC) processors A wide VLIW processor with many issue slots allows exploiting instruction-level parallelism in a thread running on that processor, while a single-

issue slot RISC processor, or a narrow VLIW processor with few issue slots, can be designed to efficiently execute a series of instructions sequentially. In practice, an application often comprises a series of threads that can be executed in parallel, where some threads are very poor in instruction-level parallelism, and some threads inherently have a large degree of instruction-level parallelism. During compilation of such an application, the application is analyzed and different threads that can be executed in parallel are identified. Furthermore, the degree of instruction-level parallelism within a thread is determined as well. This application can be mapped onto a processing system according to the invention as follows. Threads that have a large degree of instruction-level parallelism are mapped onto the wide VLIW processors, while threads that are very poor in instruction-level parallelism, or have no instruction-level parallelism at all, are mapped onto the single-issue slot RISC processors, or the narrow VLIW processors. Communication between the different threads is mapped onto the data-path connections DPC, as shown in Figure 1. As a result an efficient execution of the application is allowed: multiple threads are executed in parallel, while simultaneously instruction-level parallelism within a thread can be exploited. Therefore, a processing system according to the invention can exploit both instruction-level parallelism as well as thread-level parallelism present in an application. In addition, the present invention has the advantage of allowing for a proper match between the computational characteristics of a thread, and those of the processing element it is mapped onto. This way, an inherently sequential function like Huffman decoding is not mapped onto a wide VLIW processor, wasting architecture resources that go unused due to the lack of instruction-level parallelism, but is mapped instead onto a small RISC processor that fits its computational patterns, the wide VLIW processor remaining available for other functions.

Figure 5 shows the application graph of an application that has to be executed by a processing system shown in Figure 1. Referring to Figure 5, the application comprises five threads TA, TB, TC, TD and TE. These five threads can be executed in parallel. Threads TA, TB, TC and TE have a large degree of instruction-level parallelism, while thread TD has no instruction-level parallelism. The threads exchange data via data streams DS, and these data streams are buffered by data buffers DB. When mapping the application onto the processing system, the threads TA, TB, TC and TE are mapped each onto one of the processing elements PE17 – PE23, respectively, and thread TD is mapped onto one of the processing elements PE1 – PE15. One alternative is to map thread TA onto processing element PE17, thread TB onto processing element PE19, thread TC onto processing element

PE21, thread TD onto processing element PE15 and thread TE onto processing element PE23. In this case the threads TC, TD and TE are mapped onto processing elements that are directly connected via data-path connections DPC, i.e. processing element PE21 directly communicates with processing element PE15, and processing element PE15 directly communicates with processing element PE23. For threads TA and TB this is not the case, since processing element PE17 has to communicate with processing elements PE19 and PE21, which are indirectly coupled to PE17 via PE7 and PE9, respectively. Likewise, processing element PE19 has to communicate with processing element PE23, which is indirectly coupled to PE19 via PE11. In these cases the processing elements PE7, PE9 and PE11 can be by-passed in order to allow a direct communication between the processing elements. The data-streams DS are mapped onto data-path connections DPC, and the data-buffers DB are mapped onto a FIFO buffer BF, as shown in Figure 4. In different embodiments, the application graph may comprise more or less threads, as well as a different ratio between threads having a large degree of instruction-level parallelism and a low degree of instruction-level parallelism.

In a preferred embodiment, as shown in Figure 1, the processing elements of the first and the second set are interleaved, i.e. a processing element of the first set is arranged for direct communication with a processing element of only the second set, and a processing element of the second set is arranged for direct communication with a processing element of only the first set. As a result, there is never a penalty of more than one by-passed processing element for communication between two threads executing on different processing elements.

The degree of instruction-level parallelism and thread-level parallelism that can be exploited will vary from one application to the other, varying from applications having a low degree of thread-level parallelism wherein each thread has a high degree of instruction-level parallelism, to applications having a large degree of thread-level parallelism wherein each thread has no instruction-level parallelism. The flexibility of a processing system as shown in Figure 1 allows to map the whole range of applications onto the processing system, by by-passing processing elements onto which no thread can be mapped.

Referring to Figure 2, the interconnect network IN is a fully connected network, i.e. all functional units are coupled to all register files RF1, RF2, RF3, RF4 and

RF5. Alternatively, the interconnect network IN can be a partially connected network, i.e. not every functional unit is coupled to all register files. In case of a large number of functional units, the overhead of a fully connected network will be considerable in terms of silicon area and power consumption. During design of the VLIW processor it is decided to which degree the functional units are coupled to the register file segments, depending on the range of applications that has to be executed by the processing system.

Referring again to Figure 2, the processing element comprises a distributed register file, i.e. register files RF1, RF2, RF3, RF4 and RF5. Alternatively, the processing element may comprise a single register file for all functional units. In case the number of functional units of a VLIW processor is relatively small, the overhead of a single register file is relatively small as well.

In an alternative embodiment, the processing elements of the second set comprise a superscalar processor. A superscalar processor also comprises multiple execution units that can perform multiple operations in parallel, as in case of a VLIW processor. However, the processor hardware itself determines at runtime which operation dependencies exist and decides which operations to execute in parallel based on these dependencies, while ensuring that no resource conflicts will occur. The principles of the embodiments for a VLIW processor, described in this section, also apply for a superscalar processor. In general, a VLIW processor may have more execution units in comparison to a superscalar processor. The hardware of a VLIW processor is less complicated in comparison to a superscalar processor, which results in a better scalable architecture. The number of execution units and the complexity of each execution unit, among other things, will determine the amount of benefit that can be reached using the present invention.

In other embodiments of a processing system according to the invention, the processing system may comprise more or less processing elements than the processing system shown in Figure 1. Alternatively, the processing elements may be arranged differently, for example in a one-dimensional network, or not in an interleaved fashion, i.e. between two processing elements of the first set more than one processing element of the second set is located, and vice versa. The architecture of the processing system may depend on the range of applications that is expected to be executed on the processing system, for

example the amount of thread-level parallelism that range of applications has relative to the amount of instruction-level parallelism.

It should be noted that the above-mentioned embodiments illustrate rather than limit the invention, and that those skilled in the art will be able to design many alternative embodiments without departing from the scope of the appended claims. In the claims, any reference signs placed between parentheses shall not be construed as limiting the claim. The word "comprising" does not exclude the presence of elements or steps other than those listed in a claim. The word "a" or "an" preceding an element does not exclude the presence of a plurality of such elements. In the device claim enumerating several means, several of these means can be embodied by one and the same item of hardware. The mere fact that certain measures are recited in mutually different dependent claims does not indicate that a combination of these measures cannot be used to advantage.